Chapitre 7

Graphes valués

Ce chapitre traite deux problèmes fondamentaux de l'optimisation combinatoire. La première section est consacrée à la recherche d'un arbre couvrant de coût minimum dans un graphe valué non orienté. Nous présentons un algorithme très général fondé sur une règle des cycles et une règle des cocycles puis nous développons deux implémentations particulières conduisant aux algorithmes de Kruskal et de Prim. La seconde section décrit la recherche des chemins de coût minimum issus d'un sommet dans un graphe orienté valué. Nous exposons l'itération fondamentale de Ford, l'algorithme de Dijkstra pour des coûts positifs et sa variante A∗, l'algorithme de Bellman pour un graphe sans circuit et enfin l'algorithme Paps *lorsque aucune hypothèse particulière n'est faite sur le graphe valué.*

Introduction

La notion d'arbre de coût minimum intervient chaque fois que l'on doit relier au moindre coût des objets entre eux de telle sorte que tous les objets soient connectés directement ou non. La notion de chemin de coût minimum intervient chaque fois que l'on doit trouver un cheminement orienté d'un point à un autre au moindre coût. Ces deux problèmes sont bien résolus par des algorithmes polynomiaux efficaces qui utilisent des propriétés structurelles fortes des solutions optimales. Ces propriétés concernent les cycles et les cocycles pour le problème de l'arbre couvrant de coût minimum, les arborescences partielles pour le problème des chemins de coût minimum.

7.1 Arbre couvrant de coût minimum

7.1.1 Définition du problème

Soit $G = (S, A)$ un graphe non orienté connexe et $c : A \mapsto \mathbb{R}$ une valuation de ses arêtes. Si $H = (S, F)$ est un arbre couvrant de G, son *coût* noté $c(H)$ est défini par $\sum_{f \in F} c(f)$. Le problème est de déterminer dans l'ensemble non vide (car G est connexe) des arbres couvrants de G un arbre couvrant de coût minimum. La figure 1.1 représente un énoncé du problème qui nous servira à illustrer les propriétés et les algorithmes qui vont suivre.

Pour simplifier l'écriture, nous identifierons dans ce chapitre un graphe partiel (S, U) de G avec l'ensemble U de ses arêtes, on parlera alors du graphe partiel U.

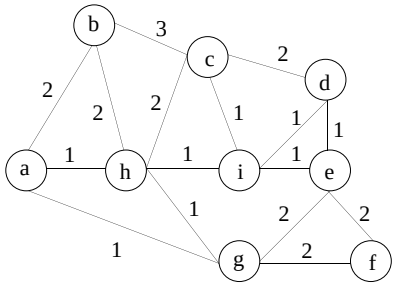

Figure 1.1: *Un graphe valué.*

7.1.2 Propriétés des arbres optimaux

On appelle *approximant* d'un arbre optimal un couple (X, Y) formé par deux sous-ensembles d'arêtes X et Y tels qu'il existe un arbre couvrant de coût minimum H contenant X et disjoint de Y. L'ensemble X contient les arêtes *admises*, l'ensemble Y contient les arêtes *écartées* et l'ensemble $Z = A - (X \cup Y)$ contient les arêtes *libres*. Etant donné un approximant (X, Y), nous allons montrer que :

a) l'existence dans G d'un *cycle candidat* ne contenant aucune arête de Y permet d'écarter une arête supplémentaire appartenant au cycle, c'est-à-dire de déterminer un meilleur approximant (X, Y') où Y' contient une arête de plus que Y.

b) l'existence dans G d'un *cocycle candidat* ne contenant aucune arête de X permet d'admettre une arête supplémentaire appartenant au cocycle, c'est-à-dire de déterminer un meilleur approximant (X', Y) où X' contient une arête de plus que X.

Une fois ces propriétés établies, la construction d'un arbre couvrant de coût minimum sera réalisée en maintenant un approximant et en l'améliorant à chaque itération. La figure 1.2 illustre les notions de cycle et de cocycle candidats pour un

approximant (X, Y) où les arêtes de X sont épaisses et les arêtes de Y en pointillé. Le cocycle $\omega(\{e\})$ est candidat, donc $X' = X \cup \{\{e, d\}\}$. Le cycle (h, b, c, h) est candidat, donc $Y' = Y \cup \{\{b, c\}\}$.

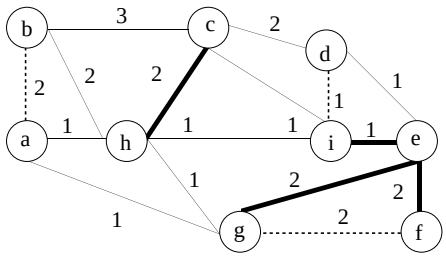

Figure 1.2: *Cycle et cocycle candidats.*

Lemme 1.1. *Soient (X, Y) un approximant, ω un cocycle de G ne contenant aucune arête de X et e une arête libre de coût minimal dans ω ; le couple $(X \cup \{e\}, Y)$ est un approximant.*

Preuve. Soit H un arbre couvrant minimal contenant X et disjoint de Y. Soit T tel que $\omega = \omega(T)$. Si ω ne contient aucune arête de X, il contient au moins une arête libre de Z car dans le cas contraire, l'arbre H ne serait pas connexe puisque ne contenant aucune arête incidente à T. Si H contient l'arête e, alors l'arbre H lui-même répond aux conditions. Sinon l'arête e ferme une chaîne élémentaire de H qui contient nécessairement une arête e' de ω distincte de e. Le graphe K obtenu à partir de H en substituant e à e' est un arbre couvrant de G de coût inférieur ou égal à $c(H)$ et donc égal à $c(H)$ puisque H est optimal. De plus K contient les arêtes de $X \cup \{e\}$ et est disjoint de Y. ∎

Lemme 1.2. *Soient (X, Y) un approximant, γ un cycle élémentaire de G ne contenant pas d'arête de Y et f une arête libre de coût maximal dans γ ; le couple $(X, Y \cup \{f\})$ est un approximant.*

Preuve. Soit H un arbre couvrant minimal contenant X et disjoint de Y. Le cycle γ ne contenant aucune arête de Y contient au moins une arête libre de Z car dans le cas contraire, l'arbre H contiendrait un cycle. Soit f l'arête libre de coût maximal de ce cycle. Si H ne passe pas par f, l'arbre H lui-même répond à la question. Sinon le graphe $H - \{f\}$ contient deux composantes connexes dont les graphes induits sont des arbres. La chaîne élémentaire obtenue à partir de γ après suppression de f contient une arête f' non contenue dans H et incidente à chacune des deux composantes connexes. Le graphe K obtenu à partir de H par substitution de f' à f est un arbre couvrant de G de coût inférieur ou égal à $c(H)$ et donc de coût égal $c(H)$ puisque H est un arbre optimal. De plus K contient X et est disjoint de $Y \cup \{f\}$. ∎

7.1.3 Algorithme général

Les deux lemmes précédents nous permettent d'énoncer deux règles, la *règle des cycles* et la *règle des cocycles*, et de construire un algorithme glouton qui détermine un arbre couvrant de coût minimum en admettant ou en rejetant un arête supplémentaire à chaque itération. Comme dans le paragraphe précédent, (X, Y) est un approximant d'un arbre optimal H.

Règle des Cycles : Soit ω un cocycle candidat ; choisir dans ω une arête libre e de coût minimal ; $Z := Z - \{e\}$; $X := X \cup \{e\}$.

Règle des Cocycles : Soit γ un cycle candidat ; choisir dans γ une arête libre e de coût maximal ; $Z := Z - \{e\}$; $Y := Y \cup \{e\}$.

L'algorithme ARBREMIN(G, c) induit par ces deux règles est alors le suivant :

```
procédure ARBREMIN(G, c) ;
   (X, Y) := (∅, ∅) ; Z := A ;
   tantque G possède un cocycle ou un cycle candidat
      appliquer la règle correspondante
   fintantque.
```

La terminaison de l'algorithme est assurée puisqu'à chaque itération l'ensemble $X \cup Y$ contient un élément de plus. Nous montrons qu'à l'issue de la dernière itération le graphe partiel X est un arbre couvrant de coût minimum.

Théorème 1.3. *L'algorithme* ARBREMIN(G, c) *détermine un arbre couvrant de coût minimum du graphe G pour la valuation c.*

Preuve. (Induction sur le numéro d'itération) Notons X_k et Y_k les sous-ensembles X et Y à l'issue de l'itération k. En utilisant les deux lemmes précédents, il est aisé de montrer par induction sur k que (X_k, Y_k) est un approximant. Notons maintenant K le numéro de la dernière itération. Si $K = m$, le graphe partiel X_K est un arbre couvrant de coût minimum car aucune arête n'est libre. Si $K < m$, il subsiste dans Z_K une arête libre e. L'arête e ne peut fermer une chaîne d'une composante connexe du graphe partiel X_K car la *règle des cycles* aurait pu être appliquée une fois de plus ; l'arête e ne peut non plus lier deux composantes connexes du graphe partiel X_K car la *règle des cocycles* aurait pu être appliquée une fois de plus. D'où la contradiction. ∎

7.1.4 Algorithmes spécifiques

Nous avons choisi de présenter, parmi les nombreux algorithmes de recherche d'un arbre couvrant de coût minimum, l'algorithme de Kruskal et l'algorithme de Prim.

Ces deux algorithmes relèvent de l'algorithme général présenté au paragraphe précédent mais utilisent chacun une stratégie spécifique pour l'ordre d'application de la *règle des cocycles* ou de la *règle des cycles*. Nous constaterons également que moyennant l'utilisation de structures de données adéquates, leur complexité dans le plus mauvais cas est assez faible.

Comme dans le paragraphe précédent, le couple (X, Y) est un approximant. Pour chacun des deux algorithmes, une itération consiste à introduire une arête supplémentaire dans X ou dans Y. Nous noterons à cet effet X_k, Y_k et Z_k les valeurs de X, Y et Z à l'issue de la $k^{ième}$ itération. Par convention nous aurons : $X_0 = Y_0 = \emptyset$ et $Z_0 = A$.

Algorithme de Kruskal

Le principe général de l'algorithme de Kruskal est d'établir une liste des arêtes ordonnée par coût croissant au sens large et d'introduire successivement chaque arête de la liste dans l'ensemble X ou dans l'ensemble Y en appliquant soit la *règle des cocycles* soit la *règle des cycles*.

```
procédure KRUSKAL(G, c);
  déterminer une liste (e_1, ..., e_m) des arêtes ordonnée par coût croissant;
  (X, Y) := (∅, ∅);
  pour i de 1 à m faire
    si e_i lie deux composantes connexes du graphe partiel X
      alors X := X ∪ {e_i}
      sinon Y := Y ∪ {e_i}
    finsi
  finpour.
```

Nous prouvons maintenant que la procédure KRUSKAL(G, c) détermine effectivement un arbre couvrant de coût minimum de G.

Théorème 1.4. *Soit $G = (S, A)$ un graphe connexe et $c : E \mapsto \mathbb{R}$ une valuation de ses arêtes, la procédure* KRUSKAL(G, c) *détermine un arbre couvrant de coût minimum de G pour la valuation c.*

Preuve. Nous montrons par induction sur le numéro d'itération k que chaque itération consiste à appliquer soit la *règle des cocycles* soit la *règle des cycles* si l'une des deux est appliquable. A l'issue de la première itération, $e_1 = \{a, b\}$ lie deux composantes connexes du graphe partiel vide d'arêtes et est une arête libre de coût minimal du cocycle $\omega(\{a\})$. La première itération applique donc la *règle des cocycles*. Supposons que (X_{k-1}, Y_{k-1}) soit un approximant. Si e_k lie deux composantes connexes C' et C'' du graphe partiel X_{k-1}, l'arête e_k est libre et de

coût minimal dans le cocycle candidat $\omega(C')$ puisque $X_{k-1} \cup Y_{k-1} = \{e_1, \ldots, e_{k-1}\}$, l'itération k applique donc la *règle des cocycles*. Sinon, l'arête e_k lie deux sommets d'une même composante connexe C du graphe partiel X_{k-1} et le graphe induit par C est un arbre. L'arête e_k ferme donc une chaîne du graphe partiel X_{k-1} et est la seule arête libre (donc de coût maximal) du cycle ainsi créé; l'itération k applique alors la *règle des cycles*. Puisque $X_m \cup Y_m = A$ et $Z_m = \emptyset$, le graphe partiel X_m est un arbre couvrant de coût minimum de G pour la valuation c. ∎

La figure 1.3 montre l'arbre couvrant (arêtes épaisses) de coût minimum calculé par l'algorithme de Kruskal.

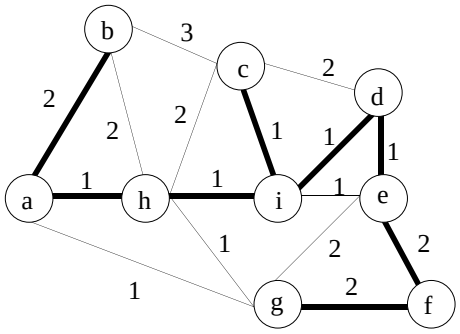

Figure 1.3: *Arbre couvrant de coût minimum.*

Implémentation de l'algorithme de Kruskal

L'algorithme de Kruskal manipule essentiellement les composantes connexes du graphe partiel X à travers les opérations de recherche de la composante connexe à laquelle appartient un sommet (chaque extrémité de l'arête e_i) et d'union de deux composantes connexes (si l'arête e_i lie ces deux composantes). Il s'agit donc de gérer ces deux opérations sur une collection d'ensembles disjoints. Une structure de données adaptée consiste à représenter chaque ensemble par une anti-arborescence dont les sommets représentent les éléments de l'ensemble et dont la racine est le représentant de l'ensemble. Si l'on note $\mathcal{V}$ cette collection, l'opérateur CRÉER$(v, \mathcal{V})$ ajoute le nouvel ensemble $\{v\}$ à $\mathcal{V}$; l'opérateur TROUVER$(v, \mathcal{V})$ détermine le représentant de l'ensemble de $\mathcal{V}$ auquel appartient v; enfin si u et v sont les représentants de deux ensembles distincts, l'opérateur UNIR$(u, v, \mathcal{V})$ remplace l'ensemble représenté par u par l'union des ensembles représentés par u et par v, et détruit l'ensemble représenté par v. La dernière section du chapitre 3 traite de l'implémentation efficace de ces opérateurs et montre en particulier que la complexité amortie d'une suite de m opérations comportant n opérations CRÉER est $O(m\alpha(m, n))$ où $\alpha(m, n)$ est une fonction réciproque de la fonction d'Ackermann.

En utilisant cette structure de données, une implémentation de l'algorithme de Kruskal est la suivante :

```
procédure KRUSKAL(G, c);
  déterminer une liste (e_1, ..., e_m) des arêtes ordonnée par coût croissant;
  {On note a_i et b_i les extrémités de e_i }
  V := ∅; X := ∅;
  pour tout s de S faire CRÉER(s, V) finpour;
  pour i de 1 à m faire
    u :=TROUVER(a_i, V); v :=TROUVER(b_i, V);
    si (u ≠ v) alors UNIR(u, v, V); X := X ∪ {e_i} finsi
  finpour.
```

La complexité du tri initial est $O(m \log n)$. Suivent alors m opérations CRÉER et au plus $2m$ TROUVER et m UNIR. Il résulte des propriétés de la fonction $\alpha(m, n)$ que la complexité globale est celle du tri des arêtes, soit $O(m \log n)$.

L'algorithme de Prim

Le principe de l'algorithme de Prim est d'appliquer $n-1$ fois la *règle des cocycles* à une suite de $n-1$ cocycles candidats associés à des sous-ensembles emboîtés de sommets. Le graphe partiel X ainsi obtenu, qui est sans cycles par construction et possède $n-1$ arêtes, est un arbre couvrant de coût minimum.

```
procédure PRIM (G, c);
  T := {a}; X := ∅; Z := A;
  {a est un sommet quelconque de S}
  pour i de 1 à n − 1 faire
    choisir une arête {x, y} de coût minimal dans ω(T);
    {on suppose : x ∈ T et y ∈ S − T}
    X := X ∪ {{x, y}};
    T := T ∪ {y}
  finpour;
  retourner(X).
```

Théorème 1.5. *Soit $G = (S, A)$ un graphe connexe et $c : A \mapsto \mathbb{R}$ une valuation de ses arêtes, la procédure* PRIM(G, c) *détermine un arbre couvrant de coût minimum de G pour la valuation c.*

Preuve. Durant son exécution, l'algorithme n'écarte aucune arête. Donc, lors de chaque itération, le cocycle $\omega(T)$ est candidat pour la *règle des cocycles* puisque G est connexe et X est inclus dans l'ensemble des arêtes du sous-graphe induit par T; l'arête $\{x, y\}$ est alors l'arête libre de coût minimal du cocycle $\omega(T)$. Chaque itération applique donc la *règle des cocycles*. ∎

La figure 1.4 montre l'arbre couvrant de coût minimum (arêtes épaisses) calculé par l'algorithme de Prim.

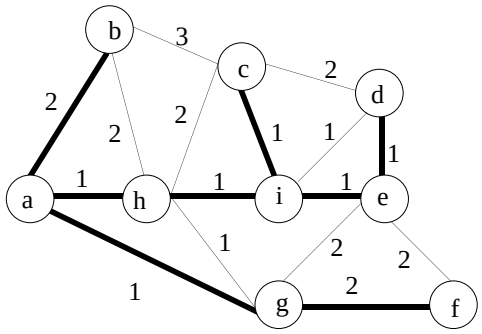

Figure 1.4: *Arbre couvrant de coût minimum.*

Implémentation de l'algorithme de Prim

L'opération fondamentale de l'algorithme de Prim est la recherche d'une arête de coût minimal d'un cocycle. Cette opération est appliquée à une suite de cocycles pour laquelle chaque sous-ensemble contient un sommet de plus que le précédent.

Une implémentation efficace de ces opérations utilise un tas (voir chapitre 3, section 3.4) pour gérer les sommets de la bordure $\mathcal{B}(T)$ de T où chaque sommet y de $\mathcal{B}(T)$ est affecté d'une priorité notée $p(y)$ égale au coût minimal d'une arête de $\omega(T)$ d'extrémité y. Un sommet de $\mathcal{B}(T)$ est *promouvable* s'il est de priorité minimale.

Nous utilisons les opérateurs suivants : Créer, Insérer, Extrairemin, Priorité, Est-élément qui sont respectivement les opérateurs de création d'un tas, d'insertion d'un élément, de suppression d'un élément de priorité minimum, d'affectation d'une nouvelle priorité à un élément et de test d'appartenance d'un élément.

Soit $\omega(T)$ le cocycle courant à l'issue d'une itération de l'algorithme de Prim et $\mathcal{T}$ le tas associé à $\mathcal{B}(T)$. Nous supposons que le nœud de $\mathcal{T}$ associé à un sommet y de $\mathcal{B}(T)$ contient outre la priorité $p(y)$ de y une arête de $\omega(T)$ d'extrémité y et de coût $p(y)$. Une arête de coût minimal de $\omega(T)$ est ainsi stockée à la racine de $\mathcal{T}$.

Soit $\{x, y\}$ l'arête sélectionnée lors de l'itération suivante ($x \in T, y \in \mathcal{B}(T)$), la mise à jour de T et de $\mathcal{B}(T)$ lors de la promotion du sommet y est réalisée par la procédure Promotion ci-dessous :

```
procédure PROMOTION(y, T);
  pour tout voisin z de y dans S − T faire
    si EST-ÉLÉMENT(z, 𝒯) alors
      si p(z) > c({y, z}) alors PRIORITÉ(z, c({y, z}), 𝒯)) finsi
    sinon INSÉRER(z, c({z, y}), 𝒯)
    finsi
  finpour;
  T := T ∪ {y}.
```

L'utlisation d'un tableau booléen permet de réaliser l'opération EST-ÉLÉMENT en temps constant. L'opération PRIORITÉ se réalise facilement si la file de priorité est implémentée par un tas. Elle prend alors un temps $O(\log n)$ où n est le nombre d'éléments du tas. Il est alors possible d'implémenter l'algorithme de Prim en utilisant la promotion comme suit :

```
procédure PRIM-AVEC-PROMOTION(G, c);
  CRÉER(𝒯);
  choisir un sommet a de S; T := {a};
  pour tout voisin v de a faire INSÉRER(v, c({a, v}), 𝒯) finpour;
  pour i de 1 à n − 1 faire
    y :=EXTRAIREMIN(𝒯);
    soit {x, y} l'arête rangée au nœud de 𝒯 contenant y;
    X := X ∪ {{x, y}};
    PROMOTION(y, T)
  finpour.
```

Chaque itération sur i supprime la racine (complexité $O(\log n)$), met X à jour (complexité $O(1)$) et réalise la promotion d'un sommet. Pour cette dernière, et dans le plus mauvais cas, il faudra exécuter pour chaque voisin le test d'appartenance au tas et soit une insertion soit une affectation de priorité. Donc le nombre total de mises à jour du tas $\mathcal{T}$ dues aux promotions est en $O(m)$. Il en résulte que la complexité globale de la procédure PRIM(G, c) est $O(m \log n)$.

7.2 Chemins de coût minimum

Lorsque les arcs d'un graphe orienté sont étiquetés par des nombres réels qui peuvent représenter des coûts, des distances, des durées ..., on définit de manière naturelle le coût d'un chemin comme la somme des étiquettes des arcs successivement empruntés par ce chemin. Un problème important est le calcul efficace d'un

chemin de coût minimum entre deux sommets donnés. Comme on peut l'imaginer aisément, ce problème intervient dans de très nombreuses applications et joue de plus un rôle central dans la résolution de certains problèmes plus difficiles tels que les flots, les ordonnancements, la programmation dynamique. Dans cette section, nous montrerons, suivant en cela la démarche unificatrice de Tarjan, que les principaux algorithmes de résolution sont issus d'une même itération fondamentale, que leur efficacité est liée à l'ordre dans lequel ces itérations sont exécutées et que pour obtenir une complexité faible, cet ordre doit dépendre des hypothèses faites sur les données. La plupart des algorithmes de résolution déterminent en fait pour tout sommet accessible à partir de l'origine un chemin de coût minimum de l'origine à ce sommet. Nous spécifierons donc le problème sous cette forme. Dans cette section, le seul algorithme présenté spécifique à un sommet origine et un sommet destination est l'algorithme appelé A^*. Il s'agit d'une variante de l'algorithme de Dijkstra qui utilise une évaluation par défaut supposée connue a priori du coût minimum d'un chemin de tout sommet au sommet destination. Cet algorithme est très utilisé pour des graphes de grande taille définis de manière implicite (problèmes de recherche en intelligence artificielle, programmation dynamique), car il permet d'obtenir la solution sans pour autant développer tous les sommets accessibles à partir de l'origine.

7.2.1 Définition du problème

Soient $G = (S, A)$ un graphe orienté, $c : A \mapsto \mathbb{R}$ une *valuation* des arcs de G et s un sommet du graphe G appelé *origine*. On suppose que tout sommet est accessible à partir de s. On appelle coût de l'arc (x, y) la valeur $c(x, y)$. Le *coût du chemin* $\gamma = (z_0, \ldots, z_q)$ est la somme $c(\gamma) = \sum_{k=1}^{q} c(z_{k-1}, z_k)$. L'objet du problème est de déterminer, pour chaque sommet x, le coût minimum noté $l(s, x)$ de s à x. La figure 2.1 représente un graphe valué muni d'un sommet origine (en grisé). Etant donnés deux chemins $\beta = (x_0, \ldots, x_p)$ et $\gamma = (y_0, \ldots, y_q)$ tels que

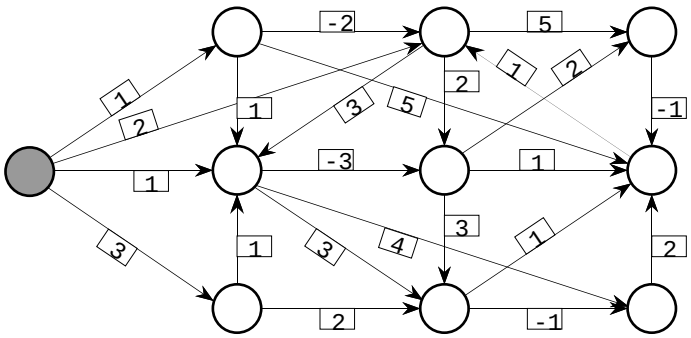

Figure 2.1: *Un graphe valué.*

$y_0 = x_p$, nous appelons concaténation des deux chemins β et γ le chemin noté $\beta\gamma$=$(x_0, \ldots, x_{p-1}, y_0, \ldots, y_q)$.

7.2.2 Existence d'une solution

La question de l'existence d'une solution repose sur la notion de *circuit absorbant*, c'est-à-dire ici de circuit de coût strictement négatif. Si un tel circuit existe, alors pour tout sommet x accessible à partir d'un sommet du circuit, il existe un chemin de s à x de coût arbitrairement petit obtenu en parcourant le circuit autant de fois que nécessaire. Le circuit $(1, 2, 3, 1)$ est absorbant pour le graphe de la figure 2.2. La proposition suivante précise la condition d'existence d'une solution.

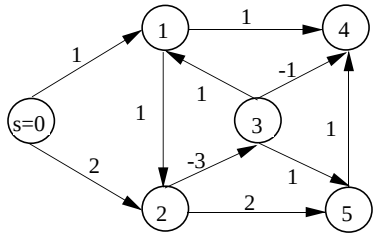

Figure 2.2: *Un circuit absorbant.*

Proposition 2.1. *Il existe un chemin de coût minimum de l'origine à tout autre sommet si et seulement si le graphe ne possède pas de circuit de coût strictement négatif.*

Preuve. La condition est bien sûr nécessaire. Réciproquement, soit x un sommet quelconque. L'ensemble des chemins *élémentaires* de s à x est dominant pour le problème car le coût d'un chemin élémentaire extrait d'un chemin γ de s à x est au plus égal au coût de γ. Les chemins élémentaires de s à x étant en nombre fini, il existe un chemin élémentaire de coût minimum. ∎

La proposition précédente n'est pas constructive et ne met pas en lumière la propriété structurelle fondamentale d'un chemin de coût minimum.

Proposition 2.2. *Soit $\gamma = (z_0, \ldots, z_q)$ un chemin de coût minimum de $x = z_0$ à $y = z_q$, tout chemin $(z_k, z_{k+1}, \ldots, z_l)$, $0 \leq k \leq l \leq q$, est un chemin de coût minimum de z_k à z_l.*

Preuve. Notons ν le chemin $(z_k, z_{k+1}, \ldots, z_l)$ et soit μ un chemin de z_k à z_l tel que $c(\mu) < c(\nu)$. Le chemin obtenu à partir de γ en remplaçant ν par μ est de coût strictement inférieur au coût de γ. Contradiction. ∎

La propriété précédente qui est élémentaire pour les chemins de coût minimum, est la base de la «programmation dynamique», une technique fondamentale pour résoudre certains problèmes d'optimisation combinatoire et de contrôle optimal. Ici, cette propriété induit une seconde condition nécessaire et suffisante fondée sur l'existence d'une arborescence partielle constituée de chemins de coût minimum. A cet effet, nous appellerons *arborescence de chemins minimaux* une arborescence partielle de G dont la racine est s et dont chaque chemin de s à x est un chemin de coût minimum de s à x dans G.

Proposition 2.3. *Si G n'a pas de circuits absorbants, G possède une arborescence des chemins minimaux.*

Preuve. Appelons *pré-arborescence des chemins minimaux* un sous-graphe $\mathcal{A} = (T, B)$ de G qui est une arborescence de racine s telle que pour tout sommet t de T, le chemin dans $\mathcal{A}$ de s à t est de coût minimum *dans* G. Nous montrons la condition par construction itérative d'une arborescence des chemins minimaux, à partir de la pré-arborescence des chemins minimaux $\mathcal{A}_0 = (\{s\}, \emptyset)$. On détermine à l'étape k une pré-arborescence des chemins minimaux $\mathcal{A}_k$ qui couvre au moins un sommet de plus que $\mathcal{A}_{k-1}$. Soient donc (voir figure 2.3)

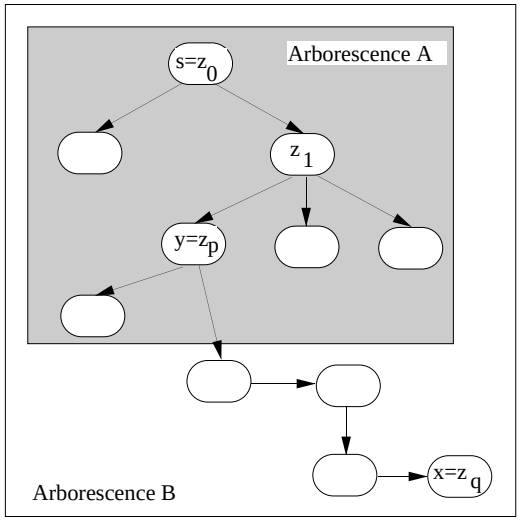

Figure 2.3: *Construction d'une arborescence des chemins minimaux.*

$\mathcal{A} = (T, B)$ une pré-arborescence des chemins minimaux telle que $S - T \neq \emptyset$,
x un sommet de $S - T$,
$\gamma = (z_0, \ldots, z_q)$ un chemin (élémentaire) de coût minimum de $s = z_0$ à $x = z_q$ dans G,
$y = z_p$ le *dernier* sommet de γ appartenant à T,
γ' le sous-chemin $(z_0, \ldots, z_p)$,
γ'' le sous-chemin $(z_p, \ldots, z_r)$ $(r \in \{p+1, .., q\})$,
β le chemin de s à y dans $\mathcal{A}$.

Les deux chemins γ' et β ont le même coût d'après la proposition 2.2. Donc, pour tout $r \in \{p+1, .., q\}$, le chemin $\beta\gamma''$ est de coût minimum dans G. De plus si $T' = T \cup \{z_{p+1}, \ldots, z_q\}$ et $B' = B \cup \{(z_j, z_{j+1}) \mid j = p, .., (q-1)\}$, alors le sous-graphe $\mathcal{A}' = (T', B')$ est une pré-arborescence des chemins minimaux qui couvre au moins un sommet de plus que $\mathcal{A}$. Il en résulte qu'après au plus $n - 1$ itérations, on obtient une arborescence des chemins minimaux. ∎

La plupart des algorithmes de recherche d'un chemin de coût minimum cherchent à construire par ajustements successifs une arborescence des chemins minimaux.

En chaque sommet x, ils font décroître une évaluation par excès $\Delta(x)$ du coût minimum d'un chemin de s à x qui est égale à la valeur d'un chemin de s à x. Le prédécesseur de x sur ce chemin, appelé *père de* x et noté $p(x)$, est également maintenu. Lors de la terminaison de l'algorithme, les arcs de l'ensemble $\{(p(x), x) \mid x \in S - \{s\}\}$ constituent une arborescence des chemins minimaux de G. Ces algorithmes utilisent comme test d'arrêt la caractérisation suivante d'une arborescence des chemins minimaux :

Proposition 2.4. *Soit $\mathcal{A}$ une arborescence partielle de racine s et soit $\lambda(x)$ le coût du chemin de s à x dans $\mathcal{A}$. L'arborescence $\mathcal{A}$ est une arborescence des chemins minimaux si et seulement si pour tout arc (x, y) de G on a : $\lambda(x) + c(x, y) \geq \lambda(y)$.*

Preuve. La condition nécessaire est immédiate. Réciproquement, soit γ un chemin de s à y. En sommant les inégalités (de la proposition) vérifiées par λ pour tous les arcs de γ, il vient : $\lambda(y) - \lambda(s) \leq c(\gamma)$ ou encore, puisque $\lambda(s) = 0 : \lambda(y) \leq c(\gamma)$. Le chemin de s à y dans $\mathcal{A}$ est donc de coût minimum dans G. ∎

7.2.3 Itération fondamentale

Soit $\Delta : S \mapsto \mathbb{R} \cup \{+\infty\}$ une fonction telle que si $\Delta(x) \in \mathbb{R}$, $\Delta(x)$ soit la valeur d'un chemin élémentaire de s à x. Un sommet x est dit *évalué* si $\Delta(x) \in \mathbb{R}$. Un arc (x, y) est dit *candidat*si $\Delta(x) + c(x, y) < \Delta(y)$ et l'on note C l'ensemble des arcs candidats. L'itération fondamentale, due à Ford, choisit un arc candidat (x, y) et exécute la mise à jour des fonctions p et Δ :

procédure AJUSTER-ARBORESCENCE(x, y) ;
$\Delta(y) := \Delta(x) + c(x, y)$; $p(y) := x$.

La figure 2.4 résume la transformation élémentaire réalisée à chaque itération.

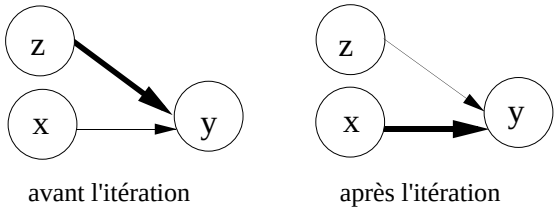

Figure 2.4: *L'itération de Ford.*

Avant cette itération on a : $\Delta(x) + c(x, y) < \Delta(y)$, $p(y) = z$ et $\Delta(z) + c(z, y) = \Delta(y)$. Après cette itération on a : $\Delta(x) + c(x, y) = \Delta(y)$, $p(y) = x$ et $\Delta(z) + c(z, y) > \Delta(y)$. La procédure AJUSTER-CANDIDATS(x, y) ci-dessous définit le nouvel ensemble d'arcs candidats.

```
procédure AJUSTER-CANDIDATS( x, y);
  C := C − {(x, y)};
  pour tout successeur z de y faire
    si Δ(y) + v(y, z) < Δ(z) alors
      C := C ∪ {(y, z)}
  finpour.
```

7.2.4 Algorithme de Ford

L'algorithme de Ford fondé sur la proposition précédente peut être implémenté par la procédure suivante :

```
procédure FORD(G, s);
  Δ(s) := 0; p(s) := ∅; C := ∅;
  pour tout sommet x de S − {s} faire Δ(x) := +∞; p(x) := ∅ finpour;
  pour tout successeur z de s faire C := C ∪ {(s, z)};
  tantque C ≠ ∅ faire
    choisir un arc (x, y) dans C;
    AJUSTER-ARBORESCENCE( x, y);
    AJUSTER-CANDIDATS( x, y)
  fintantque;
  retourner(p, Δ).
```

Le théorème suivant garantit la validité de la procédure FORD lorsque G ne possède pas de circuit absorbant.

Théorème 2.5. *Si le graphe G ne possède pas de circuit absorbant, la procédure* FORD*(G, s) se termine et à l'issue de la dernière itération, la fonction p définit une arborescence de chemins minimaux.*

Preuve. Si le graphe G ne possède pas de circuit absorbant, on montre aisément les propriétés **a)** et **b)** suivantes par induction sur le numéro d'itération :

a) si $\Delta(x)$ est fini, il existe un chemin *élémentaire* de s à x de coût $\Delta(x)$;

b) la *restriction* de la fonction p aux sommets évalués est une arborescence partielle du sous-graphe de G induit par les sommets évalués telle que pour tout sommet évalué y distinct de la racine $\Delta(p(y)) + c(p(y), y) = \Delta(y)$.

Si une itération porte sur un arc candidat (x, y) tel que y est évalué, la valeur $\Delta(y)$ décroît d'une quantité α égale à la différence (strictement positive) des coûts

de deux chemins élémentaires distincts de s à y. La valeur α est donc minorée par l'écart positif minimum ϵ entre les coûts de deux chemins élémentaires de coûts distincts. Le réel ϵ est strictement positif car l'ensemble des chemins élémentaires de G est fini, le nombre total de mises à jour de la fonction Δ pour le sommet y est donc *fini*. Il en résulte que la procédure se termine.

Lors de la terminaison, tous les sommets sont évalués, car si un sommet y ne l'était pas, il resterait au moins un arc candidat sur tout chemin de s à y. De plus, aucun arc (x, y) n'étant candidat, la fonction père définit un arborescence des chemins minimaux de G. ∎

Le théorème précédent garantit la validité de l'algorithme de Ford si le graphe G ne possède pas de circuit absorbant. Cependant, en laissant totalement libre l'ordre des itérations, c'est-à-dire le choix des arcs candidats, la convergence peut être très lente. Pour certains graphes valués à m arcs, il est possible d'exécuter jusqu'à 2^m itérations.

Pour obtenir une complexité polynomiale, la plupart des algorithmes regroupent les arcs ayant la même origine. Une itération encore appelée examen d'un sommet consiste alors à choisir un sommet et à exécuter la procédure AJUSTER-ARBORESCENCE pour tous les arcs candidats incidents extérieurement à ce sommet.

Il sera commode, pour analyser les différents algorithmes, de distinguer trois états pour un sommet. On *ouvre* un sommet x chaque fois que son évaluation décroît, on le *ferme* à l'issue de chaque exécution de EXAMINER, il reste *libre* tant qu'il n'est pas évalué. Initialement, le sommet s est ouvert et tous les autres sommets sont libres. Nous noterons respectivement O, F et L les sous-ensembles des sommets ouverts, fermés et libres. Les procédures OUVRIR(x) et FERMER(x) réalisent les mises à jour des ensembles O, F et L induites par l'ouverture et la fermeture du sommet x. La procédure INIT(G, s) ouvre le sommet s et libère tous les autres sommets.

La figure 2.5 représente l'automate des transitions possibles entre ces trois états. La procédure EXAMINER(x) implémente l'examen d'un sommet qui constitue

Figure 2.5: *Le graphe des états d'un sommet.*

maintenant une itération.

```
procédure EXAMINER(x);
  pour tout successeur y de x faire
    si Δ(x) + c(x, y) < Δ(y) alors
      AJUSTER-ARBORESCENCE( x, y); OUVRIR(y)
    finsi
  finpour;
  FERMER(x).
```

Pour obtenir un algorithme efficace, il convient de faire en sorte qu'un sommet fermé ne puisse plus (ou du moins le moins souvent possible) redevenir ouvert.

Nous allons analyser deux cas particuliers pour lesquels les itérations peuvent être ordonnées de telle sorte que tout sommet fermé reste fermé.

7.2.5 Graphe sans circuit

Soient G un graphe sans circuit et $L = (s_1, \ldots, s_n)$ une liste topologique des sommets de G telle que $s_1 = s$.

Proposition 2.6. *L'algorithme* BELLMAN(G, s) *examine les sommets une fois et une seule dans l'ordre de la liste.*

```
procédure BELLMAN(G, s);
  INIT( G, s);
  pour k de 1 à n faire EXAMINER(s_k).
```

Preuve. Lors de l'itération k ($k \geq 2$), le sommet s_k n'est pas libre car le sommet s_k (accessible à partir de s dans G) possède un prédécesseur s_i ($i < k$) placé avant lui dans la liste et qui a été examiné. Après l'exécution de EXAMINER(s_k), le sommet s_k est fermé et tous les sommets s_i ($i < k$) restent fermés puisque seuls les $\Delta(s_j)$ ($j > k$) ont pu décroître lors de cette itération. ∎

7.2.6 Algorithme de Dijkstra

Si les coûts sont positifs ou nuls, la règle qui consiste à examiner un sommet ouvert d'évaluation minimale conduit à un algorithme efficace dû à Dijkstra. L'algorithme est alors décrit par la procédure ci-dessous où INIT(G, s) initialise O, L et F.

```
procédure DIJKSTRA(G, s);
  INIT(G, s);
  répéter n − 1 fois
    choisir un sommet ouvert x d'évaluation minimale;
    EXAMINER(x)
  finrépéter.
```

La validité de cet algorithme repose sur sur la propriété suivante :

Théorème 2.7. *L'évaluation du sommet ouvert x choisi à chaque itération est le coût minimum d'un chemin de s à x dans G.*

Preuve. (Induction sur le numéro d'itération.) Le coût minimum d'un chemin de s à s est nul puisque le coût de tout circuit est positif ou nul. La propriété est donc vraie pour la première itération. Soit x le sommet ouvert choisi lors de l'itération i ($i > 1$) et F l'ensemble des sommets examinés avant l'itération i. Remarquons que d'après l'induction, tout sommet t de F a été choisi lors d'une itération antérieure à i et est resté fermé depuis en conservant une évaluation égale au coût minimum d'un chemin de s à t dans G. Il résulte alors de la procédure EXAMINER que pour tout sommet y de $S - F$, on a avant l'itération i :

$$\Delta(y) = \begin{cases} l(s,y) & \text{si } y \in F \\ \min\{l(s,z) + c(z,y) \mid z \in F \text{ et } (z,y) \in A\} & \text{si } y \in O \\ +\infty & \text{si } y \in L. \end{cases}$$

Soit $\gamma = (z_0, \ldots, z_q)$ un chemin quelconque de s à x et (z_{k-1}, z_k) le dernier arc de γ tel que $z_{k-1} \in F$ et $z_k \notin F$. On a donc avant l'itération i :

$$c(\gamma) \geq \Delta(z_{k-1}) + c(z_{k-1}, z_k) \geq \Delta(z_k) \geq \Delta(x)$$

d'après l'induction, la positivité des coûts, la relation précédente et le choix de x. Il en résulte que $\Delta(x)$ est le coût minimum d'un chemin de s à x dans G et que pour l'un des ces chemins, tous les sommets, sauf le dernier, appartiennent à F. ∎

Proposition 2.8. *L'algorithme* DIJKSTRA *sélectionne les sommets dans l'ordre croissant (au sens large) des coûts minimaux.*

Preuve. Raisonnons par induction sur le numéro d'itération et notons t le *dernier* sommet fermé avant l'itération i. Nous avons par induction pour tout sommet z de T : $\Delta(z) \leq \Delta(t)$ et $\Delta(x) \geq \Delta(t)$ puisque le sommet x n'a pas été choisi avant l'itération i . ∎

Les figures 2.6 et 2.7 décrivent une exécution de l'algorithme. Les sommets fermés sont noirs, les sommets ouverts sont gris, le sommet ouvert sélectionné est doublement cerclé, les sommets libres sont blancs et l'évaluation d'un sommet figure

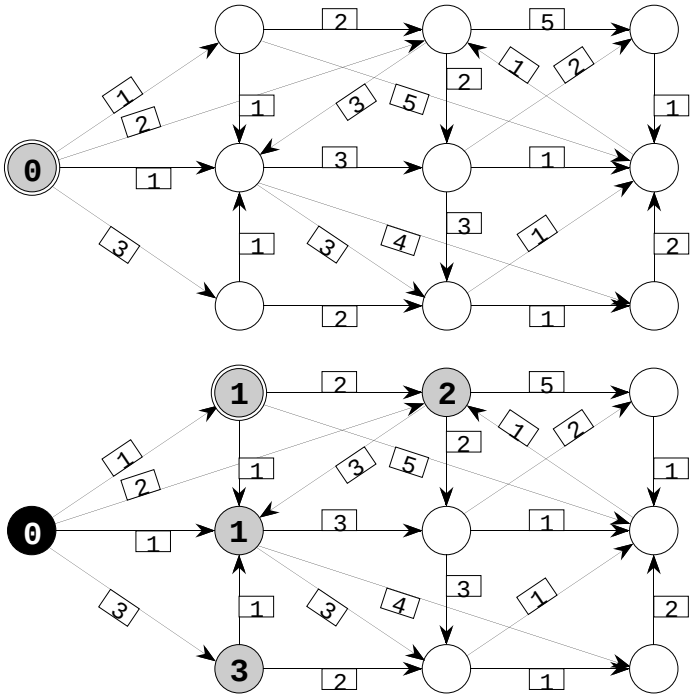

Figure 2.6: *Graphe initial et première itération.*

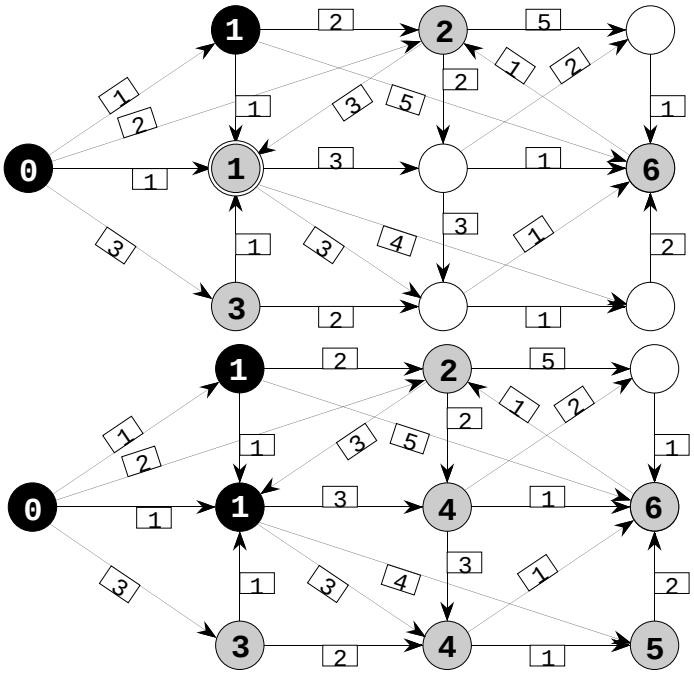

Figure 2.7: *Les deux itérations suivantes.*

à l'intérieur de ce sommet si elle est finie. L'évolution complète de l'ensemble O, du sommet sélectionné x et de la fonction Δ est donnée dans le tableau ci-dessous

(la dernière colonne est le coût minimum $l(0,x)$ de l'origine 0 au sommet x) :

O	x	Δ	
$\{0\}$	0	$\Delta(0)=0$	0
$\{1,2,3,4\}$	1	$\Delta(1)=1, \Delta(2)=1, \Delta(3)=3, \Delta(4)=2$	1
$\{2,3,4,8\}$	2	$\Delta(4)=2, \Delta(2)=1, \Delta(3)=3, \Delta(8)=6$	1
$\{3,4,8,5,6,9\}$	4	$\Delta(4)=2, \Delta(8)=6, \Delta(3)=3, \Delta(5)=4, \Delta(6)=4, \Delta(9)=5$	2
$\{3,8,5,6,9,7\}$	3	$\Delta(7)=7, \Delta(8)=6, \Delta(3)=3, \Delta(5)=4, \Delta(6)=4, \Delta(9)=5$	3
$\{8,5,6,9,7\}$	5	$\Delta(7)=7, \Delta(8)=6, \Delta(5)=4, \Delta(6)=4, \Delta(9)=5$	4
$\{8,6,9,7\}$	6	$\Delta(7)=6, \Delta(8)=5, \Delta(6)=4, \Delta(9)=5$	4
$\{8,9,7\}$	8	$\Delta(7)=6, \Delta(8)=5, \Delta(9)=5$	5
$\{9,7\}$	9	$\Delta(7)=6, \Delta(9)=5$	5
$\{7\}$	7	$\Delta(7)=6$	6

Proposition 2.9. *L'algorithme de Dijkstra détermine les chemins de coût minimum du couple (G,s) en temps $O(m \log n)$.*

Preuve. Si nous appelons *bordure* de F l'ensemble noté $\mathcal{B}(F)$ des sommets (nécessairement ouverts) de $S-F$ qui possèdent au moins un prédécesseur dans F, et si nous associons à chaque élément x de $\mathcal{B}(F)$ une priorité égale à son évaluation $\Delta(x)$, nous constatons qu'à chaque itération de l'algorithme, il faut essentiellement déterminer un élément z de $\mathcal{B}(F)$ de priorité minimale, supprimer z de $\mathcal{B}(F)$, insérer dans $\mathcal{B}(F)$ les successeurs de z qui ne lui appartiennent pas déjà et modifier éventuellement la priorité des successeurs de z qui appartiennent à $\mathcal{B}(F)$. La gestion de $\mathcal{B}(F)$ par un tas, tout à fait analogue à celle étudiée pour implémenter l'algorithme de Prim, démontre la proposition. ∎

7.2.7 Algorithme A^*.

Nous présentons dans ce paragraphe une variante de l'algorithme de Dijkstra adaptée au cas où il s'agit de déterminer dans un graphe dont les arcs sont valués par des coûts positifs ou nuls un chemin de coût minimum entre un sommet origine s et un sommet destination p.On suppose que l'on dispose initialement pour chaque sommet x d'une *évaluation par défaut* notée $h(x)$ du coût minimum $l(x,p)$ d'un chemin de x à p. L'idée fondamentale de l'algorithme A^* est alors d'associer à chaque sommet x une approximation de $l(s,p)$, notée $f(x)$, qui tient compte de la cible p par le biais de l'évaluation par défaut $h(x)$ et d'examiner en priorité un sommet ouvert d'approximation minimale.

La procédure A*(G,s,p,h) ci-dessous décrit cet algorithme.

```
procédure A*(G, s, p, h);
  INIT(G, s); f(s) = h(s); O := {s}; x := s;
  tantque x ≠ p faire
    EXAMINER*(x);
    x := sommet ouvert d'approximation f(x) minimale
  fintantque.
```

La procédure EXAMINER*(x) ne se distingue de la procédure EXAMINER(x) que par la mise à jour de l'approximation des nouveaux sommets ouverts.

```
procédure EXAMINER*(x);
  pour tout successeur y de x faire
    si Δ(x) + c(x, y) < Δ(y) alors
      AJUSTER-ARBORESCENCE( x, y);
      OUVRIR(y);
      f(y) := Δ(y) + h(y)
    finsi
  finpour;
  FERMER(x).
```

Le tableau suivant montre l'évolution de l'ensemble des sommets ouverts sur le graphe valué de la figure 2.8 pour lequel $s = 1$ et $p = 7$. Sur cette figure, l'évaluation par défaut initiale et les valeurs successives de l'approximation sont inscrites à côté de chaque sommet.

k	1	2	3	4	5	6	7
O	$\{1\}$	$\{2,3,4\}$	$\{2,4,6\}$	$\{3,4,6\}$	$\{4,6\}$	$\{4,7\}$	$\emptyset$

La validité de l'algorithme A^* repose sur le fait que l'approximation du sommet

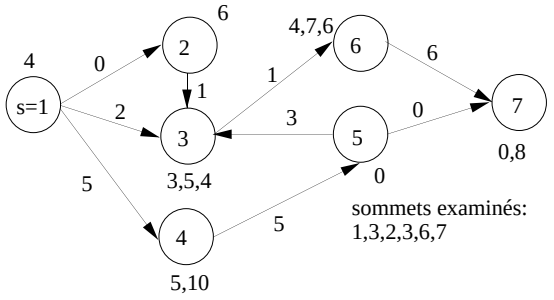

Figure 2.8: *Un graphe valué et la fonction h.*

examiné à chaque itération est une *évaluation par défaut* de $l(s,p)$. Lorsque le sommet p lui-même est examiné, son approximation vaut $f(p) = \Delta(p)$ et est donc égale à $l(s,p)$.

Proposition 2.10. *Lors de chaque itération, l'approximation $f(x)$ du sommet examiné est une évaluation par défaut de $l(s,p)$.*

Preuve. Soit $\gamma = (x_0, \ldots, x_r)$ un chemin *de coût minimum* de s à p. Nous montrons d'abord par induction qu'à l'issue de chaque itération, il existe un entier k tel que :

a) tout sommet x_i du sous-chemin $(x_0, \ldots, x_k)$ est fermé et satisfait $\Delta(x_i) = l(s, x_i)$;
b) le sommet x_{k+1} est ouvert et satisfait $\Delta(x_{k+1} = l(s, x_{k+1})$.

Remarquons d'abord que le sommet $s = x_0$ reste fermé car tout circuit passant par s est de coût positif ou nul. Il en résulte que l'invariant est vrai à l'issue de la première itération. Soit maintenant z le sommet examiné lors de l'itération i, et k l'entier associé par l'invariant à l'issue de l'itération $i-1$. Tout sommet x_i du sous-chemin $(x_0, \ldots, x_k)$ reste fermé puisque $\Delta(x_i) = l(s, x_i)$. Si $z \neq x_{k+1}$, le sommet x_{k+1} reste ouvert et l'invariant est vrai pour l'entier k à l'issue de l'itération i. Si $z = x_{k+1}$, le sommet x_{k+1} devient fermé. Considérons alors le sous-chemin $(x_{k+1}, \ldots, x_r)$ à l'issue de l'itération en cours. Ses sommets n'étant pas tous fermés puisque x_r est ouvert, notons x_j son premier sommet ouvert. Tous les sommets x_i de $(x_{k+1}, \ldots, x_{j-1})$ sont fermés et vérifient par définition $\Delta(x_i) = l(s, x_i)$. On a également $\Delta(x_j) = l(s, x_j)$ puisque le sommet x_{j-1} a été examiné. L'invariant est donc vrai pour l'entier $j-1$. L'invariant est donc conservé.

Considérons alors le sommet x_{k+1} où k est l'entier associé par l'invariant à l'itération en cours. On a :

$$f(x_{k+1}) = \Delta(x_{k+1} + h(x_{k+1}) = l(s, x_{k+1}) + h(x_{k+1}) \leq l(s,p).$$

Le sommet x examiné étant un sommet ouvert d'approximation minimale, il satisfait aussi $f(x) \leq l(s,p)$. ∎

La proposition précédente nous permet de conclure que l'algorithme A*(G, s, p, h) converge vers la solution. En effet, sa terminaison est assurée puisque son principe est l'itération de Ford et qu'il n'existe pas de circuits absorbants. De plus lorsque le sommet ouvert p est examiné, son approximation $f(p)$ satisfait $l(s,p) \geq f(p) = \Delta(p) \geq l(s,p)$.

L'algorithme A^* ne diffère de l'algorithme de Dijkstra que par le choix du sommet ouvert à examiner. En particulier si la fonction d'évalution par défaut h est nulle en tout sommet, les deux algorithmes sont identiques. De manière plus fine, on peut montrer que si la fonction h satisfait l'hypothèse de *consistance* :

$$\forall x, y \in S, \quad h(x) \leq l(s,y) + h(y)$$

alors comme dans l'algorithme de Dijkstra, le nouveau sommet examiné est celui d'évaluation minimale et donc restera fermé.

L'algorithme A^* peut être efficace dans le cas de graphes de grande taille définis de manière implicite pour lesquels on dispose d'une bonne fonction d'évaluation par défaut. En effet, dans ce cas, certains sommets ouverts dont l'approximation est de valeur élevée n'auront pas été examinés lors de la terminaison de l'algorithme.

7.2.8 Algorithme « PAPS »

Nous présentons dans ce paragraphe un algorithme pour le cas où les coûts sont des nombres réels quelconques. Sa particularité essentielle est d'examiner le sommet ouvert *le plus ancien*. Cet algorithme fournit bien entendu la solution du problème en l'absence de circuits absorbants mais permet également de déceler la présence d'un tel circuit lorsque le problème n'a pas de solution. L'ensemble des sommets ouverts est géré dans l'ordre « premier arrivé premier servi » au moyen d'une file. La procédure générale PAPS(G, s) dans laquelle la file est notée Φ implémente cet algorithme.

```
procédure PAPS(G, s);
   INIT(G, s);
   FILEVIDE(Φ); ENFILER(s, Φ);
   n(Φ) := 1; k := 0;
   tantque Φ n'est pas vide et k ≤ n − 1 faire
      ÉTAPE; k := k + 1
   fintantque;
   si Φ n'est pas vide alors afficher « circuit absorbant ».
```

La procédure ÉTAPE consiste à examiner tous les sommets présents dans la file à l'issue de l'étape précédente. On suppose que le nombre d'éléments dans la file, noté $n(\Phi)$ est maintenu lors de chaque mise à jour.

```
procédure ÉTAPE;
   pour j de 1 à n(Φ) faire
      x :=DÉFILER(Φ);
      EXAMINER(x)
   finpour.
```

Le tableau suivant montre l'évolution de l'ensemble des sommets ouverts pour le graphe valué de la figure 2.9. Sur cette figure, les évaluations successives d'un sommet sont inscrites à coté de ce sommet.

k	1	2	3	4	5	6
O	$\{0\}$	$\{1,2,3\}$	$\{5,1,6,4\}$	$\{6,5,3\}$	$\{6\}$	$\emptyset$

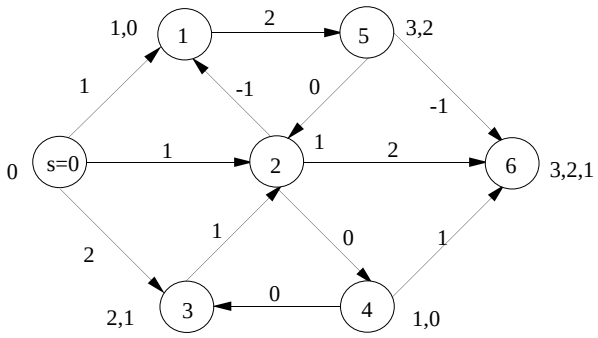

Figure 2.9: *Les évaluations de l'algorithme* PAPS.

Pour analyser l'algorithme PAPS(G, s), nous allons considérer l'ensemble noté O_k des sommets ouverts au début de l'étape k. Nous notons alors $C_k(x)$ l'ensemble (fini) des chemins de s à x de longueur inférieure ou égale à k et $\lambda_k(x)$ le coût minimum d'un chemin de $C_k(x)$ (par convention $\lambda_k(x) = +\infty$ si $C_k(x)$ est vide). L'invariant suivant est alors conservé :

Théorème 2.11. *Au début de l'étape k, l'évaluation $\Delta(x)$ d'un sommet x est égale au coût minimum d'un chemin de $C_k(x)$. Un sommet x est ouvert si et seulement si tout chemin de coût minimum de $C_k(x)$ possède exactement k arcs.*

Preuve. (Induction sur k.) La proposition est vraie par définition pour $k = 0$. Soit alors x un sommet tel que $C_{k+1}(x)$ ne soit pas vide. Au début de l'étape $k + 1$, on a :

$$\Delta_{k+1}(x) = \min\{\Delta_k(x), \min\{\Delta_k(y) + c(y, x) | y \in O_k \cap \Gamma^-(x)\}\}.$$

S'il existe un chemin de longueur inférieure ou égale à k de coût minimum dans $C_{k+1}(x)$, on a d'après l'induction $\Delta_{k+1}(x) = \Delta_k(x) = \lambda_{k+1}(x)$. Supposons maintenant que tout chemin de coût minimum dans $C_{k+1}(x)$ soit de longueur $k + 1$. Soit γ un tel chemin et (y, x) le dernier arc de γ. Tout chemin de coût minimum de $C_k(y)$ est de longueur k, donc d'après l'induction $y \in O_k \cap \Gamma^-(x)$ et $\Delta_{k+1}(x) = \lambda_{k+1}(x)$. Enfin si $C_{k+1}(x) = \emptyset$ alors $\Delta_{k+1}(x) = \lambda_{k+1}(x) = +\infty$.

Un sommet x appartient à O_{k+1} si $\Delta_{k+1}(x) < \Delta_k(x)$. Soit alors γ un chemin de coût minimum dans $C_{k+1}(x)$. Si γ est de longueur inférieure ou égale à k, nous avons $\Delta_k(x) \geq c(\gamma) = \Delta_{k+1}(x)$. Il en résulte que tout chemin de coût minimum dans $C_{k+1}(x)$ est de longueur $k + 1$. Réciproquement, si tout chemin de coût minimum de $C_{k+1}(x)$ est de longeur $k + 1$, alors, d'après l'induction, on a $\Delta_{k+1}(x) < \Delta_k(x)$ et x est bien un sommet ouvert au début de l'étape $k + 1$ (c'est-à-dire $x \in O_{k+1}$). ∎

Nous considérons maintenant le cas où l'on ne sait pas a priori si le graphe valué (G, c) contient ou non un circuit absorbant. La proposition suivante permet alors de déceler l'existence éventuelle d'un tel circuit.

Proposition 2.12. *Si au début de l'étape n, l'ensemble des sommets ouverts n'est pas vide, le graphe valué (G, c) possède un circuit absorbant.*

Preuve. D'après le théorème précédent, O_n est l'ensemble des sommets x pour lesquels tout chemin de coût minimum de $C_n(x)$ possède exactement n arcs. Si le graphe valué (G, c) ne possédait pas de circuit absorbant, il existerait un chemin *élémentaire* de coût minimum de s à x et donc de coût minimum dans $C_n(x)$. ∎

Le tableau suivant montre l'évolution de l'ensemble des sommets ouverts pour le graphe valué de la 2.10. Le circuit $(3, 2, 4, 3)$ est absorbant.

k	0	1	2	3	4	5	6	7
O	$\{0\}$	$\{1,2,3\}$	$\{5,1,6,4\}$	$\{6,5,3\}$	$\{2,6\}$	$\{1,4\}$	$\{3,5\}$	$\{2,6\}$

La complexité d'une étape quelconque de l'algorithme PAPS(G, s) est en $O(m)$

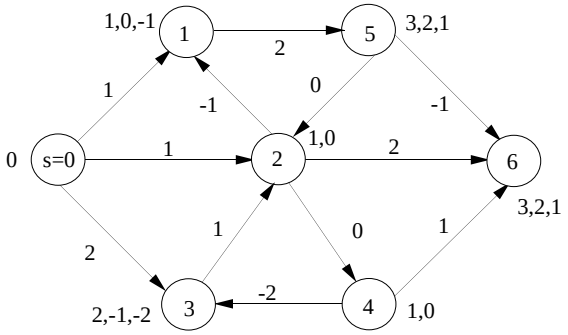

Figure 2.10: *L'algorithme* PAPS *avec un circuit absorbant.*

puisque un sommet est examiné au plus une fois. Comme l'algorithme exécute au maximum n étapes, sa complexité est $O(mn)$.

Notes

La présentation unifiée des algorithmes concernant les arbres couvrants de coût minimum par l'emploi de la *règle des cocycles* et de la *règle des cycles* est due à R.E. Tarjan. Il en est de même de la présentation des algorithmes de recherche des chemins de coût minimum en tant que variantes de l'algorithme «générique» de Ford :

R.E. Tarjan, *Data Structures and Network Algorithms*, SIAM Philadelphia, 1983, chapitres 6 et 7, 71–95.

Les algorithmes sur les arbres couvrant de coût minimum sont assez anciens. Par contre leur complexité a plus récemment été améliorée par l'utilisation de structures de données adaptées comme la gestion des partitions d'un ensemble pour l'algorithme de Kruskal et les files de priorité pour l'algorithme de Prim.

Si l'on restreint l'ensemble des arbres couvrants admissibles en ajoutant des contraintes supplémentaires, le problème devient en général beaucoup plus difficile. Si, par exemple, on limite le degré des sommets, le problème est alors NP-difficile sauf si le degré d'un seul sommet est limité.

Les algorithmes de recherche de chemins de coût minimum ont été très étudiés. Ils interviennent en effet de manière centrale dans beaucoup d'autres domaines comme les flots ou les ordonnancements. Ajouter des contraintes au problème de base le rend souvent beaucoup plus complexe. Par exemple, rechercher un chemin élémentaire de coût minimum entre deux sommets d'un graphe valué quelconque est un problème NP-difficile.

Exercices

7.1. Soit $G = (S, A)$ un graphe non orienté connexe valué par une fonction coût $c : A \mapsto \mathbb{R}$. Soit $H = (S, B)$ un graphe partiel non connexe de G. On appelle *arête de liaison* entre deux composantes connexes C et C' de H un arête qui a l'une de ses extrémités dans C et l'autre dans C'. Soit C une composante connexe de H, $\omega(C)$ le cocycle associé à C et e une arête de coût minimal de $\omega(C)$.
a) Démontrer qu'il existe un arbre couvrant de coût minimum de G qui contient e.
On considère l'algorithme de Sollin décrit par la procédure ci-dessous :

```
procédure SOLLIN(G, c);
  H := (S, ∅);
  tantque H n'est pas connexe faire
    soient C_1, ..., C_p les composantes connexes de H;
    {les C_k ne sont pas marquées au départ};
    tantqu'il existe une composante connexe C_k non marquée faire
      marquer(C_k);
      soit e = {x, y} une arête de coût minimum dans ω(C_k);
      {x est l'extrémité de e qui appartient à C_k}
      soit C_l la composante connexe contenant y;
      marquer(C_l);
      ajouter l'arête e à H
    fintantque
  fintantque.
```

b) Démontrer que l'algorithme précédent applique la *règle des cocycles* à chaque itération de la boucle *tantque* interne.
c) En déduire que lors de la terminaison, H est un arbre couvrant de coût minimum.

7.2. Soit $G = (S, A)$ un graphe orienté complet à n sommets et valué par une fonction coût $c : A \mapsto \mathbb{N}$ telle que :

1. $\forall x, y \in S, \quad c(x,y) = c(y,x)$;
2. $\forall x, y, z \in S, \quad c(x,y) + c(y,z) \geq c(x,z)$.

Un *tour* $t = (t_1, \ldots, t_n)$ de G est une permutation de S et le coût $c(t)$ du tour t est la somme :

$$c(t_n, t_1) + \sum_{k=1}^{n-1} c(t_k, t_{k+1}).$$

On note G' la version non orientée de G où le coût $c'(x,y)$ de l'arête $\{x,y\}$ est égal à $c(x,y)$. Soient H un arbre couvrant de coût minimum de G', $P' = (p_1, \ldots, p_n)$ un parcours en profondeur de G' à partir du sommet s et t le tour de G associé à P'. On note $c'(H)$ le coût de l'arbre H et $c^*(G)$ le coût minimum d'un tour de G.
a) Démontrer que $c^*(G) > c'(H)$.
b) Démontrer que $c(t) \leq 2c'(H)$.
c) Obtient-on le même résultat avec un parcours en largeur de H ?

7.3. On considère le problème de transport monoproduit défini par m fournisseurs F_i et n clients C_j. Le fournisseur F_i dispose d'un stock $a_i \in \mathbb{N}$, le client C_j a une demande $b_j \in \mathbb{N}$ et l'on suppose que $\sum a_i = \sum b_j$. Le coût *unitaire* de transport du fournisseur F_i au client C_j est un entier naturel c_{ij}. Le graphe valué G associé au problème a pour sommets les m fournisseurs et les n clients et ses arcs sont les couples (F_i, C_j) valués par c_{ij}. Un plan de transport $X = (x_{ij})$ est une solution du système :

$$\begin{aligned} \sum_{j \in \{1,\ldots,n\}} x_{ij} &= a_i \quad i \in \{1, \ldots, m\} \\ \sum_{i \in \{1,\ldots,m\}} x_{ij} &= b_j \quad j \in \{1, \ldots, n\} \\ x_{ij} \geq 0 \quad i \in \{1, \ldots, m\} &\quad j \in \{1, \ldots, n\} \end{aligned}$$

et le coût $c(X)$ du plan X est $\sum_{i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\}} c_{ij} x_{ij}$. La matrice du système est notée M.
a) Démontrer qu'il existe un plan de transport optimal entier.
b) Démontrer que le rang de M est $m + n - 1$.

On appelle *solution de base* un ensemble de $m + n - 1$ colonnes indépendantes de M.

c) Montrer que le graphe partiel des arcs (F_i, C_j) associés à ces colonnes est un arbre couvrant de G. La réciproque est-elle vraie ?

Soit H un arbre couvrant de coût minimum de G. Soit $B(H)$ la solution de base associée à H (si elle existe).

d) Montrer qu'il existe un seul plan de transport tel que $x_{ij} = 0$ pour $(F_i, C_j) \notin B(H)$. Montrer que ce plan n'est pas nécessairement optimal.

7.4. Déterminer un graphe valué à m arcs muni d'un sommet origine s tel que l'algorithme « générique » de Ford réalise dans le plus mauvais cas $O(2^m)$ itérations.

7.5. Soit $G = (S, A)$ un graphe, soit s un sommet origine et soit $c : A \mapsto \mathbb{N}$ une fonction capacité. La capacité d'un chemin est la plus petite capacité des arcs de ce chemin. Adapter l'algorithme de Dijkstra à la recherche des chemins de capacité maximale de s à tous les autres sommets de G.

7.6. On considère l'algorithme A^* présenté dans la section 7.2.6. Démontrer que si la fonction d'évaluation par défaut h satisfait :

$$\forall (x, y) \in A, \quad h(x) \leq c(x, y) + h(y),$$

un sommet examiné restera fermé jusqu'à la terminaison de l'algorithme.

7.7. On considère l'algorithme «générique» de Ford et l'on note E l'ensemble des sommets examinés. Démontrer que les invariants suivants sont conservés à chaque itération :

> si $\Delta(x)$ est fini, il existe un chemin *élémentaire* de s à x de coût $\Delta(x)$;
> la *restriction* de la fonction père aux sommets de E est une arborescence partielle du sous-graphe de G induit par E telle que pour tout sommet y de E distinct de la racine $\Delta(p(y)) + c(p(y), y) = \Delta(y)$.

7.8. On appelle *graphe conjonctif* un graphe orienté $G = (S, A)$ valué par $c : A \mapsto \mathbb{R}$, muni d'une racine α et d'une anti-racine ω, et tel qu'il existe pour tout sommet x un chemin positif ou nul de α à x et de x à ω. Un *ensemble de potentiels* d'un graphe conjonctif G est une application $t : S \mapsto \mathbb{R}$ satisfaisant $t(\alpha) = 0$ et :

$$\forall (x, y) \in A, \quad t(y) - t(x) \geq c(x, y)$$

On note $T(G)$ l'ensemble des ensembles de potentiels de G.
a) Démontrer que $T(G)$ est non vide si et seulement si G ne possède pas de circuit strictement positif.
On suppose maintenant que $T(G)$ n'est pas vide et l'on note $r(x)$ le coût *maximum* d'un chemin de α à x.

b) Démontrer que r est le plus petit élément de $T(G)$ pour la relation d'ordre :

$$t \leq t' \Longleftrightarrow \forall x \in S, \quad t(x) \leq t'(x).$$

On considère l'ensemble $D(G)$ des ensembles de potentiels tels que $t(\omega) = r(\omega)$.

c) Démontrer que la fonction f définie par $f(x) = r(\omega) - C(x, \omega)$ où $C(x, \omega)$ est le coût maximum d'un chemin de x à ω est le plus grand élément de $D(G)$.

7.9. Soit $G = (S, A)$ un graphe orienté valué par $c : A \mapsto \mathbb{Q}$. On note $S = \{1, \ldots, n\}$, $S_k = \{1, \ldots, k\}$, G_k le graphe induit par S_k et l'on suppose que $s = 1$ est un sommet origine pour tous les sous-graphes G_k. On note enfin $\alpha_k(i)$ la valeur minimale d'un chemin de 1 à i dans G_k *passant au plus une fois par le sommet* k. On considère alors l'algorithme de Rao décrit ci-dessous :

```
procédure RAO(G, c);
(1)  k := 1; R := ∅; abs := 0;
(2)  si c(1,1) < 0 alors abs := 1
(3)  sinon
(4)     tantque abs = 0 et k ≤ n faire
(5)        k := k + 1; R := R ∪ {k}; T := {k};
(6)        α_k(k) := min{α_{k-1}(j) + c_{jk} | j ∈ Γ⁻_{G_k}(k)};
(7)        tantque abs = 0 et T ≠ R faire
(8)           U(T) := ω⁺_{G_k}(T);
(9)           si U(T) ≠ ∅ alors
(10)             calculer d_pq = min{d_ij | (i,j) ∈ U(T)}
                 {où d_ij = α_k(i) + c(i,j) − α_{k-1}(j)}
(11)          finsi;
(12)          si U(T) = ∅ ou d_pq > 0 alors
(13)             pour tout i dans R − T faire
(14)                α_k(i) := α_{k-1}(i); T := T ∪ {k}
(15)             finpour
(16)          sinon
(17)             α_k(q) := α_k(p) + c(p,q); T := T ∪ {q};
(18)             si (q,k) ∈ A et α_k(q) − α_k(k) + c(q,k) < 0 alors
(19)                afficher(« circuit absorbant » );abs := 1
(20)             finsi
(21)          finsi
(22)       fintantque
(23)    fintantque
(24) finsi.
```

a) Démontrer que la valeur de $\alpha_k(k)$ calculée à la ligne (6) est bien le coût minimum d'un chemin de 1 à k dans G_k passant au plus une fois par k.

b) Démontrer que si $U(T)$ est vide ou si d_{pq} est positif ou nul, le coût minimum d'un chemin de 1 à i dans G_k passant au plus une fois par k est $\alpha_{k-1}(i)$.

c) Démontrer que si d_{pq} est strictement négatif, alors $\alpha_k(p) + c(p,q)$ est le coût minimum d'un chemin de 1 à q dans G_k passant au plus une fois par k.

d) En déduire que l'algorithme RAO détermine les chemins de coût minimum de 1 à tous les autres sommets de G ou détecte la présence d'un circuit absorbant.